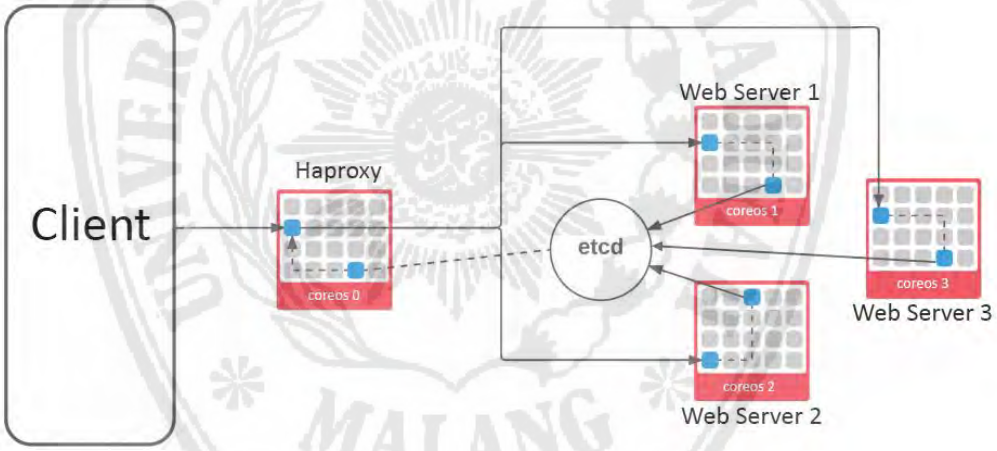
BAB III
PERANCANGAN DAN PEMBUATAN SISTEM

3.1. Perancangan Sistem

Untuk merancang sistem ini diperlukan 3 buah *web server* dan 1 buah *server* untuk *load balance*. *Server-server* ini berada pada jaringan lokal yang sama. Untuk mengakses *web server*, *client* diharuskan untuk melewati *server load balance* terlebih dahulu. *Server load balance* akan mmemilih *web server* mana yang dipilih untuk melayani *request* dari *client*. Untuk memilih *web server* tersebut, *server load balance* menggunakan algoritma Round Robin dan parameter yang dapat menentukan pembagian beban. Topologi perancangan sistem ini dapat dilihat pada gambar 3.1.



Gambar 3.1 Skema perancangan Cluster CoreOS.

3.2. Alat dan Bahan

Perancangan sistem *load balancing* ini membutuhkan minimal 4 PC untuk dapat diimplemetasikan. Instalasi *hardware* maupun *software* utuk menyusun sistem ini sangatlah diperlukan Untuk meyusun rancangan sistem ini, kami menggunakan *hardware* dan software sebagai berikut:

- Hardware

  Node 1

  Processor : Intel (R) Xeon (R) CPU E31220 @ 3.10 GHz

Harddisk : 1 TB HDD

Memory : 4 GB DDR2

Node 2

Processor : Intel (R) Xeon (R) CPU X3430 @ 2.40 GHz

Harddisk : 500 GB HDD

Memory : 4 GB DDR2

Node 3

Processor : Intel(R) Xeon(R) CPU E5405 @ 2.00 GHz

Harddisk : 500 GB HDD

Memory : 4 GB DDR2

Load Balance Server

Processor : Pentium (R) Dual Core CPU E5400 @ 2.70 GHz

Harddisk : 320 GB HDD

Memory : 4 GB DDR2

- *Software*

Node 1

Sistem Operasi : Linux Core OS

Cluster : etcd

Web server : Nginx

Node 2

Sistem Operasi : Linux Core OS

Cluster : etcd

Web server : Nginx

Node 3

Sistem Operasi : Linux Core OS

Cluster : etcd

Web server : Nginx

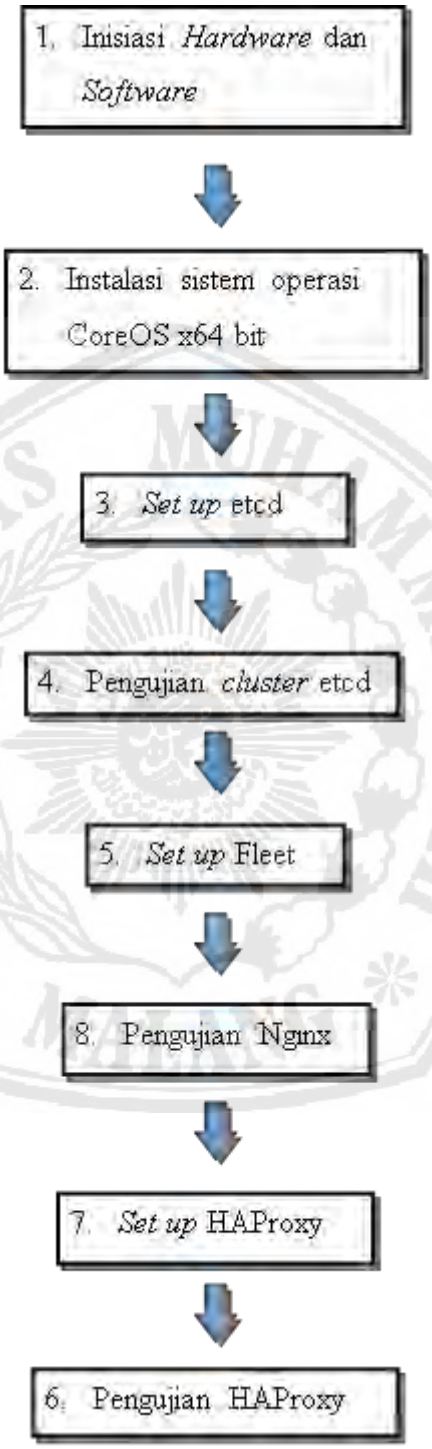
Load Balance Server

Sistem Operasi : Linux Core OS

Load Balance : HAProxy

3.3. Alur Penelitian

Tahap-tahap dalam melakukan percobaam ini disajikan dalam bagan alur penelitian seperti berikut.



Gambar 3.2 Bagan alur penelitian.

Penjelasan gambar 3.2 :

1. Inisiasi *hardware* dan *sofware*

   Tahap ini merupakan tahap awal / persiapan sebelum merujuk ke instalasi aplikasi. Menyiapkan selurh kebutuhan *hardware* dan *software* yang dibutuhkan dalam implementasi *cluster*ing di CoreOS.

2. Instalasi sistem operasi CoreOS

   Pada tahap ini, semua *hardware* yang dibutuhkan harus sudah terpenuhi dari rancangan sistem yang sudah *fix* untuk melanjutkan instalasi sistem operasi CoreOS. Instalasi ini dilakukan pada ketiga PC atau node.

   a. Pertama *download image* CoreOS ISO di https://coreos.com/os/docs/latest/booting-with-iso.html pilih yang versi stabil.

   b. Langkah berikutnya adalah untuk me-*mount image* CoreOS ISO yang kita miliki. Kemudian boot up mesin virtual CoreOS.

Gambar 3.3 Tampilan awal CoreOS.

   c. Membuat *user Login*

   Ada dua hal penting untuk dicatat di sini. Pertama, kita masih belum menginstal versi *standalone* CoreOS. Apa yang kita miliki sekarang ini mirip dengan *live* CD dari OS berjalan. Kedua, *user* core ini otomatis log in untuk pertama kalinya, tetapi kita harus membuat *user* lain untuk *Login* ketika CoreOS *boot up* waktu berikutnya.

   Jadi, mari kita pertama kali membuat akun pengguna dan password, menggunakan *file cloud-config*. Hal ini deilakukan dengan perintah "*sudo openssl passwd -1 > cloud-config.yml*" kemudian isi password untuk *Login*.

Gambar 3.4 Proses pembuatan *password user* CoreOS.

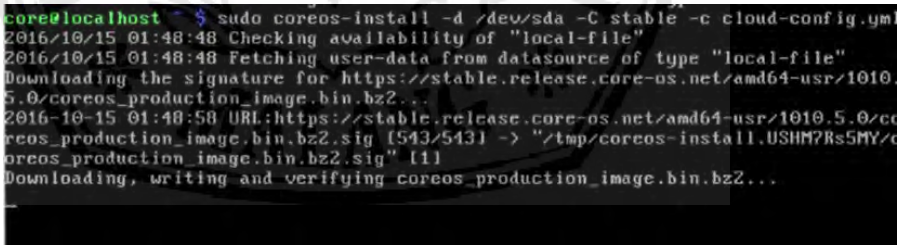
d. Edit *file* "*cloud-config*.yml"

Kita edit *file* ini untuk nemambahkan nama *user* serta parameter-parameter yang dapat dicakup oleh *user* tersebut. Untuk mengedit, kami menggunakan perintah "*vi cloud-config.yml*" seperti yang terlihat pada gambar 3.5.

Gambar 3.5 Penambahan parameter pada user CoreOS.

e. Mengunduh CoreOS *stable*

Langkah instalasi akhir adalah untuk mengunduh dan menginstal *image* CoreOS pada VirtualBox. Yaitu dengan mengetikkan perintah "*sudo coreos-install -d /dev/sda -C stable -c cloud-config.yml*".



Gambar 3.6 Mengunduh CoreOS stable.

f. *Shutdown* CoreOS

Setelah proses mengunduh selesai, matikan CoreOS dengan perintah "*sudo shutdown –h now*".

Gambar 3.7 Mematikan CoreOS.

3. *Set up* etcd

   Pada tahap ini dilakukan penganturan antar node agar dapat berkomunikasi dengan baik. Dan *file* dapat tersinkronisasi.

   a. Membuat *discovery token*

      CoreOS menggunakan etcd untuk menghubungkan mesin/node dalam *cluster*. Selain itu, etcd memilih pemimpin (*leader*) dalam *cluster* secara otomatis. Hanya satu nide yang bertugas sebagai *leader*, sedangkan yang lain adalah pengikut (*worker*). Namun *worker*-pun dapat menjadi *leader* jika *leader down* karena masalah hardware atau apa pun alasannya.

      Untuk mendapatkan *discovery token* sanagat mudah. Kita hanya perlu mengakses halaman *url* https://discovery.etcd.io/new?size=3, dengan nilai '*size*' bergantung pada berapa jumlah node yang hendak dibuat. Pada percobaan kali ini, penulis menggunakan 3 node.

   b. Mengatur ulang *cloud-config*

      Setelah itu kita masukkan *discovery token* yang telah diperoleh tadi pada *cloud-config*. Karena CoreOS menggunakan *cloud-config* sebagai konfigurasi parameter untuk mesin dan *service*, menjalankan *systemd units* secara otomatis saat *system booting*.

Gambar 3.8 Penambahan parameter *cloud config* pada node 1

Gambar 3.9 Penambahan parameter *cloud config* pada node 2 (*worker*).

Gambar 3.10 Penambahan parameter *cloud config* pada node 3 (*worker*).

4. Testing *cluster* etcd

   Dilakukan ujicoba atas pengaturan yang telah dilakukan sebelumnya. Disini akan terlihat apakah pengaturan etcd telah berfungsi dengan baik atau belum. Pengujian sendiri dilakukan dengan cara membuat sebuah *folder* dan *file* di node 1, kemudian diakses melalui node 2 dan node 3.

5. *Set up* fleet

   Tahap ini dilakukan pengaturan agar *service* dapat dijalankan dalam *cluster*. Fleet adalah manajer *cluster* yang mengotrol systemd pada level

cluster. Untuk menjalankan *service* pada *cluster*, kita harus menjalankan *systemd units regular* yang dikombinasikan dengan beberapa *fleet-spesific properties*.

a. Lakukan pemeriksaan mesin custer yang ada.

```
core@coreos1 ~ $ fleetctl list-machines
MACHINE          IP              METADATA
ac036b0e...      10.10.102.182   -
c1fb55d9...      10.10.102.183   -
c50809a0...      10.10.102.181   -
core@coreos1 ~ $
```

Gambar 3.11 Pemeriksaan mesin yang tercluster.

b. Membuat *file service* pada host.

```
[Unit]
Description=Nginx web server %i
After=docker.service
Requires=docker.service

[Service]
TimeoutStartSec=0
EnvironmentFile=/etc/environment
ExecStartPre=-/usr/bin/docker kill nginx%i
ExecStartPre=-/usr/bin/docker rm nginx%i
ExecStart=/usr/bin/docker run -v /webserver:/usr/share/nginx/html \
--name nginx%i -h nginx%i \
-p ${COREOS_PRIVATE_IPV4}:80:80 nginx
ExecStop=/usr/bin/docker stop nginx%i

[X-Fleet]
Conflicts=nginx@*.service
```

Gambar 3.12 Membuat *file service* pada host.

c. Memasukkan *file service* agar dapat dieksekusi oleh fleet.

```
core@coreos1 ~ $ fleetctl submit nginx@.service
Unit nginx@.service inactive
core@coreos1 ~ $ fleetctl list-unit-files
UNIT            HASH    DSTATE          STATE           TARGET
nginx@.service  f16227e inactive        inactive        -
core@coreos1 ~ $
```

Gambar 3.13 Memasukkan *file service* agar dapat dieksekusi oleh fleet.

d. Mendistribusikan *file service* ke semua node.

```
core@coreos1 ~ $ fleetctl load nginx@{1..3}.service
Unit nginx@1.service inactive on ac036b0e.../10.10.102.182
Unit nginx@2.service inactive
Unit nginx@3.service inactive
Unit nginx@1.service loaded on ac036b0e.../10.10.102.182
Unit nginx@2.service loaded on c1fb55d9.../10.10.102.183
Unit nginx@3.service loaded on c50809a0.../10.10.102.181
```

Gambar 3.14 Mendistribusikan *file service* ke semua node.

e. Melihat *file service* yang telah di distribusikan pada semua node.

```
core@coreos1 ~ $ fleetctl list-units
UNIT            MACHINE                         ACTIVE          SUB
nginx@1.service ac036b0e.../10.10.102.182       inactive        dead
nginx@2.service c1fb55d9.../10.10.102.183       inactive        dead
nginx@3.service c50809a0.../10.10.102.181       inactive        dead
```

Gambar 3.15 Melihat *file service* yang ada semua node.

f. Menjalankan *service* ke kesemua node.

```
core@coreos1 ~ $ fleetctl start nginx@{1..3}.service
Unit nginx@1.service launched
Unit nginx@3.service launched on c50809a0.../10.10.102.181
Unit nginx@2.service launched on c1fb55d9.../10.10.102.183
```

Gambar 3.16 Menjalankan *file service* ke semua node.

g. Melihat unit yang sedang berjalan.

```
core@coreos1 ~ $ fleetctl list-units
UNIT            MACHINE                         ACTIVE  SUB
nginx@1.service ac036b0e.../10.10.102.182       active  running
nginx@2.service c1fb55d9.../10.10.102.183       active  running
nginx@3.service c50809a0.../10.10.102.181       active  running
```

Gambar 3.17 Melihat unit/node yang sedang berjalan.

6. Pengujian fleet

   Pengujian ini dilakukan dengan cara menjalankan *balance* nginx pada *cluster*.

7. *Set up* HAProxy

   Tahap ini dilakukan konfigurasi HAProxy. Harus dikonfigurasi dengan teliti, apabila ip address yang dikonfigurasi salah, maka *request* dari *client* tidak akan pernah direspon oleh *web server*.

a. Membuat file haproxy.cfg

Pertama buat terlebih dahulu file haproxy.cfg di host coreos0 (*server load balance*), pada diretory "/haproxy".

```
frontend Server_Lokal
        bind    *:80
        default_backend Web_Server_ku
        option  forwardfor

backend Web_Server_ku
        balance roundrobin
        server  nginx1.local 10.10.102.181:80 check
        server  nginx2.local 10.10.102.182:80 check
        server  nginx3.local 10.10.102.183:80 check
```

Gambar 3.18 Isi file haproxy.cfg.

b. Menguduh HAProxy

Unduh *images* HAProxy pada coreos0 dengan perintah "docker pull haproxy:1.7". Karena penulis disini menggunakan HAProxy versi 1.7.

```
core@coreos0 ~ $ docker pull haproxy:1.7
1.7: Pulling from library/haproxy

6d827a3ef358: Pull complete
b40da44b9cf6: Pull complete
00b1fddea0c6: Pull complete
cb22d2f68860: Pull complete
99613a28a85c: Pull complete
Digest: sha256:24266f42f90f12565de7e5f54535f5a910d01906261aa1c782638e7b5ac8e29f
Status: Downloaded newer image for haproxy:1.7
```

Gambar 3.19 Mengunduh *images* HAProxy 1.7.

c. Periksa *images* HAProxy yang sudah di unduh

Pastikan *image* HAProxy telah terunduh dengan cara "*docker image*".

```
core@coreos0 ~ $ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
haproxy             1.7                 4aa0a1d3cada        6 hours ago         135.6 MB
```

Gambar 3.20 Melihat *images* yang telah terunduh.

d. Jalankan *images* HAProxy

Setelah dipastikan *images* HAProxy telah terunduh, kemudian jalankan *images* tersebut dengan perintah "*docker run --network host --name haproxy –v /haproxy:/usr/local/etc/haproxy:ro –d haproxy:1.7*".

```
core@coreos0 ~ $ docker run --network host --name haproxy -h haproxy \
> -v /haproxy:/usr/local/etc/haproxy:ro -d haproxy:1.7
de7204cc81bf1533138b90586444783fbb063c621cae54650b8959aad61cc905
```

Gambar 3.21 Menjalankan *images* HAProxy.

Penjelasan gambar 3.21

- --network host = Menjalankan *container* pada jaringan host (coreos0).
- --name haproxy = Nama *container* 'haproxy'.
- -h haproxy = Menggunakan nama hostsname 'haproxy'.
- -v /haproxy:/usr/local/etc/haproxy:ro = di gunakan untuk mount file konfigurasi HAProxy.
- -d = Menjalankan *container* sebagai background dan mencetak id.
- haproxy:1.7 = Merupakan nama *images*.

e. Periksa *container* HAProxy

Periksa apakah HAProxy sudah berjalan dengan perintah "*docker ps*".

```
core@coreos0 ~ $ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
de7204cc81bf        haproxy:1.7         "/docker-entrypoint.s"   18 seconds ago      Up 17 seconds                           haproxy
```

Gambar 3.22 Melihat *container* yang sedang berjalan.

8. Pengujian HAProxy

Tahap trahir untuk pengujian ini adalah *load balancing*. Apabila *request* dari *client* di respon oleh *web server* berati konfigurasi dari awal sampai akhir benar.